To read the article online, visit http://www.4GuysFromRolla.com/articles/121405-1.aspx

Examining ASP.NET's Membership, Roles, and Profile - Part 2

By Scott Mitchell

Introduction

In Part 1 of this article series we saw how ASP.NET's membership service provides a framework for managing user accounts. The framework is composed of a `Membership` class with a bevy of methods that can be used to create, delete, modify, retrieve, and authenticate users. Since every developer's needs are different, a concrete membership framework would be virtually useless except for new, simple projects. Thankfully the `Membership` class is designed using the *provider model*, meaning that the membership framework's actual implementation can be customized. ASP.NET ships with two membership providers - `SqlMembershipProvider` and `ActiveDirectoryMembershipProvider` - and you can build your own, if needed.

Many websites that provide user accounts need to group users in various *roles*. The roles a user belongs might specify what web pages they have access to, what information they see on the screen, and whether or not certain regions in a page are editable or view-only. Grouping users into roles and basing functionality and authorization based on a user's role is quite easy in ASP.NET thanks to the roles service. Like the membership service, the roles service defines a framework for programmatically creating and deleting roles, assigning and removing roles from users, and determining what users belong to a role, and to what roles a user belongs.

In this article we will examine ASP.NET's role service. We'll start by seeing how to setup and configure the roles service on a website, along with how to base authorization rules using roles. In addition, we'll look at how to programmatically work with the roles service, and see how to use the LoginView Web control to show information based on the logged in user's role. Read on to learn more!

First Things First...

In order to have your website support roles you first need to have your site provide user accounts. These accounts could be serialized in the database through the `SqlMembershipProvider`, or stored in an Active Directory. Regardless, roles are a concept that only applies to sites that has users to which roles can be assigned! If you're unfamiliar with ASP.NET's membership service, please take a moment to read Part 1 of this article series before tackling this part...

Setting Up Your Website to Support Roles

Like the membership service, the roles service was designed using the provider model. ASP.NET ships with three implementations of role providers:

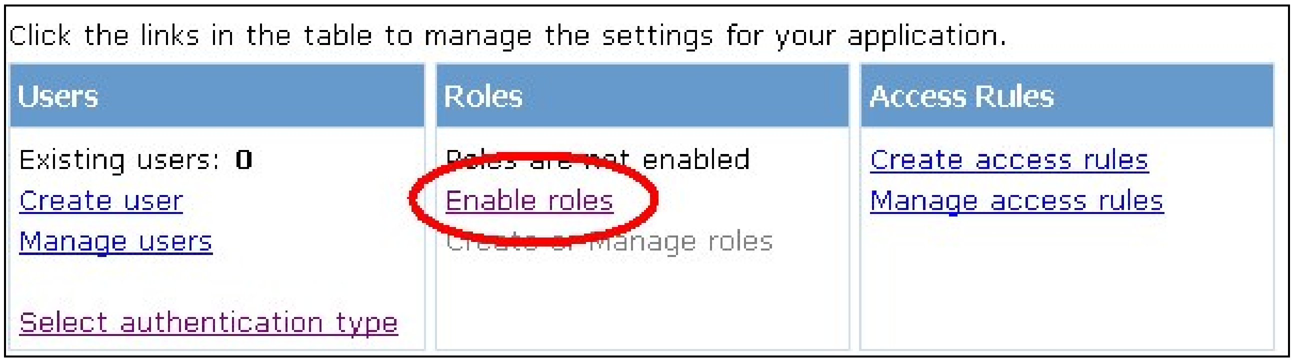
- **`SqlRoleProvider`** (the default) - stores role information in a SQL Server database. If you are using the `SqlMembershipProvider`, use this provider for roles. Specifically, the roles service uses two tables: `aspnet_Roles`, which has a record for each role in the system, and `aspnet_UsersInRoles`, which associates uses from the `aspnet_Users` table with roles from the `aspnet_Roles` table.
- **`WindowsTokenRoleProvider`** - gets group information from the Windows user store. If you are authenticating visitors using Windows authentication, this provider will allow you to view the groups to which the the visiting user belongs.
- **`AuthorizationStoreRoleProvider`** - provides role information from an authorization-manager policy

store, such as Active Directory.

In this article we will examine only the `SqlRoleProvider` provider. In order to use this provider we must have the appropriate database tables created. As we saw in Part 1 of this article series, this can be accomplished in one of two ways:

- Through the ASP.NET SQL Server Registration Tool (`aspnet_regsql.exe`) - this tool enables you to specify a SQL Server 2000 or SQL Server 2005 database to which the required database tables, views, and stored procedures will be copied.
- Through the ASP.NET Website Administration Tool - when setting the authentication type to "From the internet," the Website Administration Tool automatically calls the `aspnet_regsql.exe` tool, creating the appropriate database tables in the `ASPNETDB.mdf` SQL Server 2005 database in the site's `App_Data` folder.

Assuming you have created the needed database tables, either indirectly through the ASP.NET Website Administration Tool or directly through the `aspnet_regsql.exe` tool, to enable the role service go to the ASP.NET Website Administration Tool, click on the Security tab, and click the "Enable roles" link, as shown in the screenshot below:


Click the links in the table to manage the settings for your application.

Users	Roles	Access Rules
Existing users: 0 Create user Manage users Select authentication type	Roles are not enabled Enable roles Create or Manage roles	Create access rules Manage access rules

Doing this simply adds the following line to your site's `Web.config` file:

```
<roleManager enabled="true" />
```

If you are using a custom database to store the SQL database for the `SqlMembershipProvider` and `SqlRoleProvider` providers, you'll need to specify a provider in the `<roleManager>` element like so:

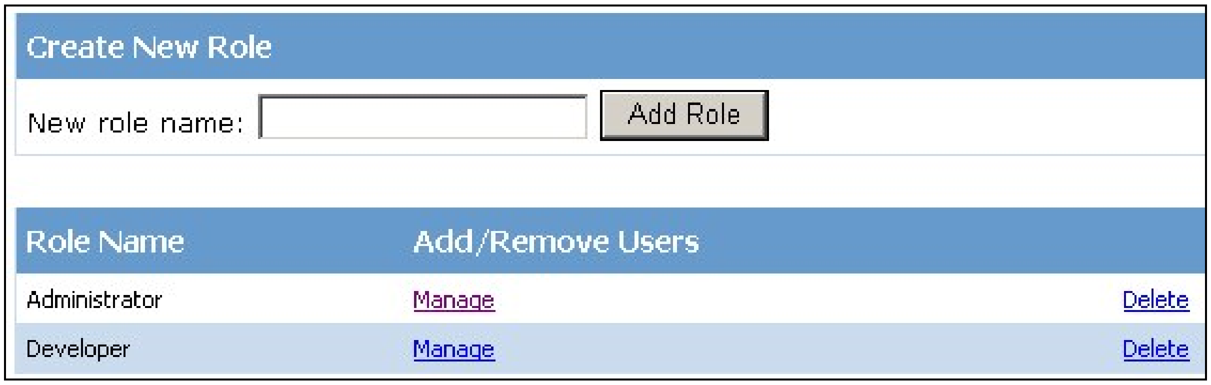
```
<configuration>
  <connectionStrings>
     <add name="MyDB" connectionString="..." />
  </connectionStrings>
  <system.web>
    ... authentication & authorization settings ...

    <roleManager enabled="true"
                 defaultProvider="CustomizedRoleProvider">
      <providers>
         <add name="CustomizedRoleProvider"
              type="System.Web.Security.SqlRoleProvider"
              connectionStringName="MyDB"
              applicationName="/" />
      </providers>
    </roleManager>
  </system.web>
</configuration>
```

Where the `<connectionString>` setting `myDB` has the connection string to the SQL Server database where the membership and role database tables reside.

The `applicationName` setting provides the name of the application using the roles system and should be set to the same `applicationName` setting used by the Membership provider. This setting should *always* be set. See Scott Guthrie's blog entry Always set the "applicationName" property when configuring Membership and other providers for more information. (There are additional attributes you can specify in both the `<roleManager>` and `<providers>` elements to further customize the roles service. We'll examine these in a future article in the series.)

Managing Roles

The role service provides a `Roles` class that has methods for creating and deleting roles, and adding and removing users from roles. These methods are useful if you want to programmatically perform these tasks, but often your site will have a set number of roles, and you want to manually specify what users belong to what roles. This can all be accomplished through the ASP.NET Website Administration Tool. Once roles are enabled, the "Enable roles" link shown previously changes to two links: "Disable roles" and "Create or Manage roles." Clicking the "Create or Manage roles" takes you to a screen where you can add new roles and manage or delete existing roles.



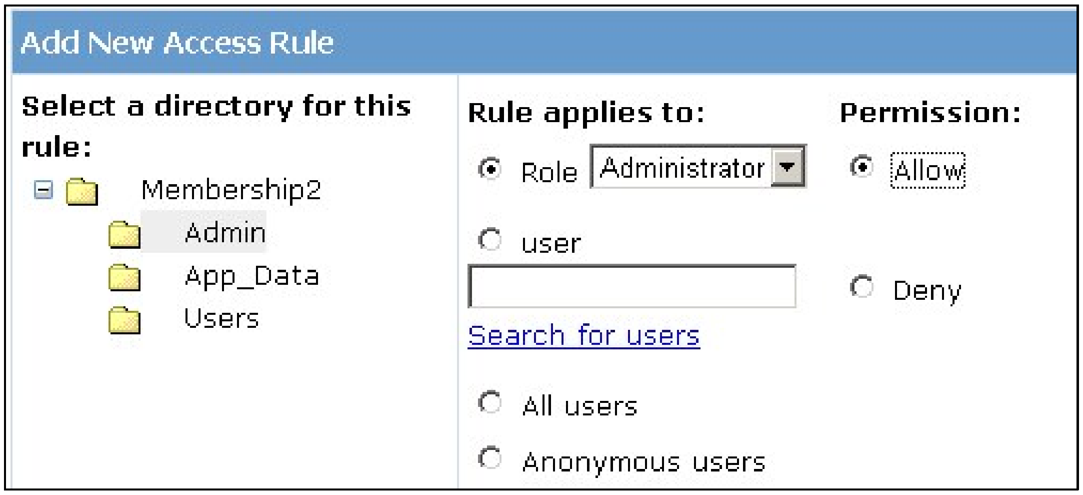
Role Name	Add/Remove Users	
Administrator	Manage	Delete
Developer	Manage	Delete

Clicking on the Manage link allows you to associate a set of users with the role. Similarly, back on the Security tab you can instead choose to "Manage users," from where you can associate a set of roles with a particular user.

Role-Based Authorization

In ASP.NET version 1.x URL-based authorization could be specified on both the user and role levels using the `<authorization>` element in the `Web.config` file. Specifically, an `<authorization>` element can contain multiple `<allow>` and `<deny>` elements that dictate a folder or URL's authorization rules. By default, *anyone* can access a URL; therefore, in order to limit access you need to use the right combination of `<allow>` and `<deny>` elements. Refer to Authorizing Users and Roles for more information on ASP.NET version 1.x's authorization syntax and semantics.

The concepts and syntax for user- and role-based authorization are the same for ASP.NET version 2.0. One nice thing about 2.0, though, is that these authorization settings can optionally be specified through the ASP.NET Website Administration Tool. (You can still manually enter the `<authorization>` element in `Web.config` if you so choose.) From the Website Administration Tool's Security tab, click on the "Create access rules" link. This will take you to a screen where you can apply authorization settings for users, user classes (`*` for all users, `?` for anonymous users), or roles (if roles are supported in the system). Simply select a folder to which the rules apply, specify the user, user class, or role, and then choose the access rights (Allow or Deny).


Add New Access Rule

Select a directory for this rule:

- Membership2
  - Admin
  - App_Data
  - Users

Rule applies to:

[✓ Role] Administrator

user

Search for users

All users

Anonymous users

Permission:

[✓ Allow]

Deny

Working with the `Roles` Class

The `Role` class in the .NET Framework provides programmatic access to the role service through a bevy of static methods. The following is a handful of the methods in this class:

- `CreateRole(`*`roleName`*`)` - adds a new role to the system.
- `DeleteRole(`*`roleName`*`)` - deletes a role from the system.
- `AddUserToRole(`*`userName`*`, `*`roleName`*`)` - adds a particular user to a particular role.
- `IsUserInRole(`*`roleName`*`)` / `IsUserInRole(`*`userName`*`, `*`roleName`*`)` - returns true or false, depending if the currently logged in user or the user specified is in the specified role.
- `GetAllRoles()` - returns a string array of all of the roles in the system.
- `GetRolesForUser()` / `GetRolesForUser(`*`userName`*`)` - returns a string array of all of the roles to which either the currently logged in user or the user specified belongs.

These methods can be used to display the roles in the system on a web page, or list the roles a user belong to, or determine whether or not a user belongs to a particular role. The examples included at the end of this article provide two pages that illustrate using the `Roles` class in code: `UserList.aspx`, which lists all users in the system along with the roles to which they belong; and `RoleList.aspx`, which lists the roles in the system along with the users that belong to each role.

Displaying Role-Specific Markup With the LoginView Control

In Part 1 of this article series we took a brief look at the myriad of new security-related Web controls in ASP.NET 2.0. One such control was the LoginView control, which provides two templates: `AnonymousTemplate` and `LoggedInTemplate`. When an unauthenticated visitor arrives at the page, they are shown the markup within the `AnonymousTemplate`; when a logged in user visits, they see the contents of the `LoggedInTemplate` instead.

The LoginView control can contain additional role-based templates. To specify a role-specific template, add to the LoginView control a `<RoleGroups>` element with `<asp:RoleGroup Roles="`*`comma-delimited list of roles`*`">` templates inside, like so:

```
<asp:LoginView ID="LoginView1" runat="server">
     <AnonymousTemplate>
         OMG, you are, like so not logged in!
     </AnonymousTemplate>

     <RoleGroups>
         <asp:RoleGroup Roles="Developer">
             <ContentTemplate>
                 Welcome back! You are a Developer, I can
                 tell by your svelte figure and impeccable
```

```
                social skills!
            </ContentTemplate>
        </asp:RoleGroup>
        <asp:RoleGroup Roles="Administrator">
            <ContentTemplate>
                We all bow down to our system adminstrators!
            </ContentTemplate>
        </asp:RoleGroup>
    </RoleGroups>

    <LoggedInTemplate>
        You are logged in!! But, wait, you are not
        a member of any roles.
    </LoggedInTemplate>
</asp:LoginView>
```

When a visitor arrives at this page, if they are not authenticated they'll see the markup in the `AnonymousTemplate`. If they are logged in, but don't belong to any roles, they'll see the `LoggedInTemplate`. If, however, they belong to a role, they'll see that role's appropriate template instead.

Realize that the LoginView control only displays *one* template. So if you are logged in a a Developer you'll see just the Developer template, and **not** the `LoggedInTemplate`. Similarly, if you are in both the Developer and Administrator roles, you'll only see the role template that comes first in the list (Developer, in this case).

Conclusion

As we saw in this article, ASP.NET 2.0's support for roles is similar to that of membership: both services are built upon the provider model, both expose a single class with a number of static methods for accomplishing tasks, and both can be configured through the ASP.NET Website Administration Tool. Of course, these similarities are not terribly surprising, seeing as roles are just an abstraction on top of user accounts...)

With roles enabled, you can create and assign roles either programmatically using the `Roles` class or manually through the Website Administration Tool. ASP.NET allows for role-based authorization rules, so you can limit access to various files and folders on a role-by-role basis. Finally, the LoginView control provides role-based templates, meaning that the output shown on a page can differ based on the user's role.

Happy Programming!

- By Scott Mitchell

Attachments

- Download the code used in this article

Article Information	
Article Title:	ASP.NET.Examining ASP.NET's Membership, Roles, and Profile - Part 2
Article Author:	Scott Mitchell
Published Date:	December 14, 2005
Article URL:	http://www.4GuysFromRolla.com/articles/121405-1.aspx